

# Программирование Python

Файлы и файловая система

Кафедра ИВТ и ПМ

2018



# План

Прошлые темы

Введение

Основные операции по работе с файлами

Файлы. Текстовый файлы  
Comma-Separated Values

Менеджеры контекста

Типы bytes и bytearray

Работа с файловой системой



# Outline

Прошлые темы

Введение

Основные операции по работе с файлами

Файлы. Текстовый файлы  
Comma-Separated Values

Менеджеры контекста

Типы bytes и bytearray

Работа с файловой системой



# Прошлые темы



# Outline

Прошлые темы

**Введение**

Основные операции по работе с файлами

Файлы. Текстовый файлы  
Comma-Separated Values

Менеджеры контекста

Типы bytes и bytearray

Работа с файловой системой



**Файл** - именованная область на диске.

**Текстовый файл** - файл, содержащий текстовые данные.

**Бинарный(двоичный) файл** - последовательность произвольных байтов.

# Бинарный файл

Фрагмент бинарного файла , переставленный в виде шестнадцатеричных значений каждого байта.

```
5089 474e 0a0d 0a1a 0000 0d00 4849 5244
0000 8007 0000 3804 0608 0000 e800 c1d3
0043 0000 7304 4942 0854 0808 7c08 6408
0088 0000 7419 5845 5374 666f 7774 7261
0065 6e67 6d6f 2d65 6373 6572 6e65 6873
```

Одному байту соответствует двузначное шестнадцатеричное число.

Порядок байт - little-endian.



# Текстовые и бинарный файлы

Текстовые файлы противопоставляются бинарным, однако являются видом последних.

Разница между бинарным и текстовым файлом только в подходах к представлению информации.

В текстовом файле информация представлена в виде последовательности символов. Одному символу может соответствовать один или несколько байт.

В бинарных файлах информация представлена непосредственно значениями байт.





# Текстовые и бинарный файлы

Например число 123 будет представлено тремя символами в текстовом файле. Для хранения этих трёх символов понадобится три байта.

В бинарном файле для хранения этого значения нужно будет задать значение одного байта.

# Текстовые файлы

## Преимущества

- ▶ Простота чтения и редактирования человеком

## Недостатки

- ▶ В общем случае - избыточный способ представления информации



# Бинарные файлы

## Преимущества

- ▶ Отсутствие избыточности

## Недостатки

- ▶ Не доступен для чтения человеком



# Outline

Прошлые темы

Введение

Основные операции по работе с файлами

Файлы. Текстовый файлы  
Comma-Separated Values

Менеджеры контекста

Типы bytes и bytearray

Работа с файловой системой



Так как разница между текстовыми и бинарными файлами только в способе представления информации, то методы работы с ними будут совпадать с точностью до записитения данных.



## Операции с текстовыми файлами

- ▶ **open('имя файла', режим)** - открыть файл. Возвращает файловый объект (файловую переменную)

```
f = open('myfile.txt', 'rt')
```

аналогично

```
f = open('myfile.txt')
```



## Некоторые режимы

- ▶ **r** - чтение (по умолчанию)
- ▶ **w** - запись
- ▶ **+** - чтение и запись
- ▶ **a** - запись в конец файла
  
- ▶ **t** - открыть как текстовый файл (по умолчанию)
- ▶ **b** - открыть как бинарный файл

Полное описание режима включает способ работы с файлом (чтение и запись) и способ представления файла (текстовый или бинарный)



# Файлы

Файловая переменная не содержит в себе файл, а только содержит общую информацию о файле и том способе как он был открыт.

Также в неявном виде в файловой переменной содержится указатель на текущую позицию, где остановилось чтение или запись.

Файловая переменная (файловый объект) содержит следующие атрибуты:

- ▶ **file.closed** - True если файл был закрыт.
- ▶ **file.mode** - режим доступа, с которым был открыт файл.
- ▶ **file.name** - имя файла.
- ▶ **file.softspace** - False если при выводе содержимого файла следует отдельно добавлять пробел.





```
f = open('myfile111')
```

Если указанного файла (открывается для чтения) в текущем каталоге не существует, то программа аварийно завершится с ошибкой:

FileNotFoundError: No such file or directory: 'myfile111'



# Файлы

```
f = open('very_important_file', 'w')
```

Если файл уже существует и открывается для записи, то всё его содержимое будет уничтожено.

# Текстовые файлы

Операции с текстовыми файлами

- ▶ `f.close()` - закрыть файл.

Перед закрытием в файл будут записано содержимое файлового буфера.

После того как файл был закрыт чтение или запись невозможны.



## Файловый буфер

Если после записи файл не закрыть (и не вызывать метод flush), то часть записанных данных может не попасть в файл.

Операция обращения к диску медленная по сравнению к обращению к ОЗУ.

Для ускорения работы с файлами вместо обращения к диску при каждой операции записи данные записываются сначала в специальный буфер (**файловый буфер**) расположенный в оперативной памяти. Такой механизм называется **кэшированием**.

После того как буфер заполнен, его содержимое записывается в файл. Таким образом минимизируется число обращений к диску, но возникает необходимость по окончании работы с файлом освободить файловый буфер.



# Outline

Прошлые темы

Введение

Основные операции по работе с файлами

**Файлы. Текстовый файлы**  
Comma-Separated Values

Менеджеры контекста

Типы bytes и bytearray

Работа с файловой системой



# Операции с файлами

- ▶ **read()** - читает весь файл. Возвращает содержимое файла.

Если файл текстовый, то возвращает строку.

Если файл бинарный, то возвращает значение типа bytes (байтовую строку)

Если файл пуст или указатель стоит в конца файла, то метод возвращает пустую строку (или пустую строку байт)

Работа с данными, которые находятся в оперативной памяти, происходит быстрее, однако всегда стоит помнить о том, что загружаемый файл может иметь большой объём и занять всю доступную оперативную память или существующую её часть.



# Операции с файлами

- ▶ **read(size)** - читает и возвращает size байт.

Если файл текстовый, то возвращает строку.

Если файл бинарный, то возвращает значение типа bytes (байтовую строку)

При каждом вызове read указатель с файле смещается на указанное число байт.

При чтении многострочного файла стоит помнить, что каждая строка заканчивается символом конца строки (обозначается "\n")



# EOF и EOL

В документации для обозначения конца файла обычно используют аббревиатуру **EOF (end-of-file)** - конец файла.

А для обозначения конца строки **EOL (end-of-line)**.

Если указатель в файле стоит в его конце, то говорят что он указывает на EOF.





# Операции с файлами

- ▶ **tell()** -> возвращает текущую позицию
- ▶ **seek(n, whence = 0)** - смещает позицию на N байт и возвращает новую позицию в файле

whence = 0 - смещение от начала файла

whence = 1 - смещение от текущей позиции

whence = 2 - смещение с конца файла



# Операции с файлами

Если открыть файл, и переместить указатель в его конец, то в size будет записан размер файла в байтах<sup>1</sup>.

```
f = open('my-file')  
f.seek(0, os.SEEK_END)  
size = f.tell()  
f.close()
```

---

<sup>1</sup>Для получения размера файла лучше использовать функцию `os.path.getsize()`



# Операции с файлами

- ▶ **readlines()** - читает и возвращает список из всех строк файла.
- ▶ **readlines(n)** - читает и возвращает список из n строк файла.  
n может быть больше общего числа строк в файле.

Если читать больше нечего, то возвращает пустой список.

- ▶ **readline()** - читает одну строку из файла.

При чтении файла построчно указатель в файле смещается соответственно и всегда будет указывать либо на начало новой строки либо на конец файла.



# Операции с файлами

- ▶ **write( mystring )** - записывает любую строку в открытый файл.

Метод `write` не добавляет символ переноса строки.

См. описание других методов в документации по файлам.



## Пример 1

Чтение файла целиком и вывод на экран по частям

```
f = open("test.txt", "r")
for line in f:
    print(line)
f.close()
```

Для файлов небольшого объёма стоит предпочитать подход, в котором сначала считывается всё содержимое, а потом происходит обработка таких данных.

Это минимизирует количество обращений к диску, а значит ускорит работу программы, однако несколько увеличит объём используемой оперативной памяти.



## Пример 2

```
f = open('my_file', 'w')  
f.write('01234567890123456789')  
f.seek(5)  
f.write('Hello, World!')  
f.close()
```

```
f = open('my_file')  
f.read()
```

Каким будет содержимое файла?



## Пример 2

```
f = open('my_file', 'w')  
f.write('01234567890123456789')  
f.seek(5)  
f.write('Hello, World!')  
f.close()
```

```
f = open('my_file')  
f.read()
```

Каким будет содержимое файла?

01234Hello, World!89



# Outline

Прошлые темы

Введение

Основные операции по работе с файлами

Файлы. Текстовый файлы  
Comma-Separated Values

Менеджеры контекста

Типы bytes и bytearray

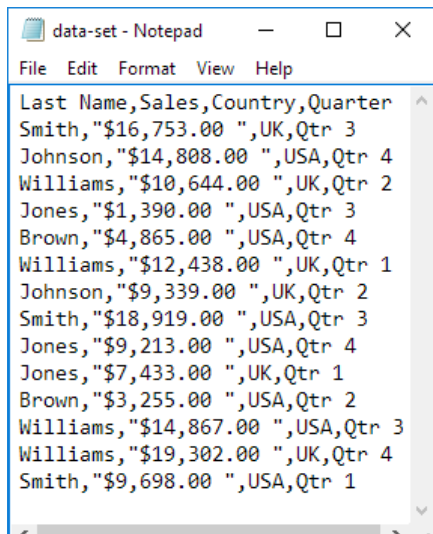
Работа с файловой системой





## Comma-Separated Values

**CSV** (Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных.



```
data-set - Notepad
File Edit Format View Help
Last Name,Sales,Country,Quarter
Smith,"$16,753.00 ",UK,Qtr 3
Johnson,"$14,808.00 ",USA,Qtr 4
Williams,"$10,644.00 ",UK,Qtr 2
Jones,"$1,390.00 ",USA,Qtr 3
Brown,"$4,865.00 ",USA,Qtr 4
Williams,"$12,438.00 ",UK,Qtr 1
Johnson,"$9,339.00 ",UK,Qtr 2
Smith,"$18,919.00 ",USA,Qtr 3
Jones,"$9,213.00 ",USA,Qtr 4
Jones,"$7,433.00 ",UK,Qtr 1
Brown,"$3,255.00 ",USA,Qtr 2
Williams,"$14,867.00 ",USA,Qtr 3
Williams,"$19,302.00 ",UK,Qtr 4
Smith,"$9,698.00 ",USA,Qtr 1
```



# Comma-Separated Values

## Спецификация:

- ▶ Каждая строка файла — это одна строка таблицы.
- ▶ Разделителем (англ. delimiter) значений колонок является символ запятой (,). Однако на практике часто используются другие разделители, то есть формат путают с DSV<sup>2</sup> и TSV<sup>3</sup>.
- ▶ Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками ("). Если в значении встречаются кавычки — они представляются в файле в виде двух кавычек подряд.

---

<sup>2</sup>delimiter-separated values — значения разделённые разделителем

<sup>3</sup>tab separated values — значения, разделённые табуляцией



# Comma-Separated Values

CSV файлы легко импортируются в excel и другие табличные редакторы.

Text Import Wizard - Step 2 of 3

This screen lets you set the delimiters your data contains. You can see how your text is affected in the preview below.

**Delimiters**

- ☐ Tab
- ☐ Semicolon
- ☒ Comma
- ☐ Space
- ☐ Other:

☐ Treat consecutive delimiters as one

Text qualifier:

**Data preview**

product_id	product_parent_id	manufacturer_id	supplier_id	product_on_sale	price
1	0	1	1	0	0
2	0	1	2	0	0
3	0	2	2	0	0
4	0	2	2	0	0

# Comma-Separated Values

CSV файлы - это де факто стандарт для хранения данных в текстовом виде.

CSV часто используются для хранения научных, статистических и данных о финансах.

Для их визуализации используется **gnuplot** - программа для создания двух- и трёхмерных графиков.

Для работы с csv файлами в python используется модуль csv. Если требуется производить статистическую обработку данных то лучше использовать модуль pandas.



## Пример

Требуется загрузить данные о деревьях для дальнейшей обработки.

CSV файл:

```
"Index", "Girth (in)", "Height (ft)", "Volume(ft^3)"
```

```
1, 8.3, 70, 10.3
```

```
2, 8.6, 65, 10.3
```

```
3, 8.8, 63, 10.2
```

```
4, 10.5, 72, 16.4
```



## Пример

```
f = open('trees.csv')
f.readline()
lines = f.readlines()

# информация о деревьях
# список из (номер, обхват, высота, объём)
data = []
for line in lines:
    t = line.split(',')
    if len(t)==4:
        data += [ (int(t[0]), float(t[1]), \
                    float(t[2]), float(t[3])) ) ]

for d in data:
    print(d)
f.close()
```



# Outline

Прошлые темы

Введение

Основные операции по работе с файлами

Файлы. Текстовый файлы  
Comma-Separated Values

Менеджеры контекста

Типы bytes и bytearray

Работа с файловой системой



# Менеджеры контекста

## Традиционный подход

```
fp = open("./file.txt", "w")  
fp.write("Hello, World")  
fp.close()
```

## С использованием менеджера контекста

```
with open("./file.txt", "w") as fp:  
    fp.write("Hello, World")
```





# Outline

Прошлые темы

Введение

Основные операции по работе с файлами

Файлы. Текстовый файлы  
Comma-Separated Values

Менеджеры контекста

Типы `bytes` и `bytesarray`

Работа с файловой системой



# Тип bytes

Тип **bytes** похож на строковый тип (**str**), где вместо отдельных символов - строки.

bytes - неизменяемый тип.

Создание переменной типа bytes

- ▶ из строки:

```
bs = b'bytes'
```

Такой способ создания байтовой строки возможен только для ASCII символов.



# Тип bytes

## Создание переменной типа bytes

- ▶ из строки с указанием кодировки<sup>4</sup> <sup>5</sup>

```
bs = bytes("Питон", "cp1251")
```

- ▶ из строки

```
b = "йцукен".encode()
```

---

<sup>4</sup> в Python используются строки в Unicode кодировке (обозначение utf-8)

<sup>5</sup> не Юникод кодировки использовать не рекомендуется потому, что они не универсальны



# Тип bytes

## Создание переменной типа bytes

- ▶ строка с закодированными значениями

```
bytes(b'\xcf\xe8\xf2\xee\xed')
```

шестнадцатеричное значение каждого байта приводится после префикса \x

- ▶ из списка целых чисел. каждый элемент списка задаёт значение одного байта.

```
bs = bytes([1,2,3])
```



# Тип bytes

Доступ к отдельным байтам производится по индексам

```
b = "й12цукен".encode()
```

```
b[0]    # 208
```

```
b[1]    # 49
```

```
b[2]    # 50
```



# Тип bytearray

**bytearray** - изменяемый bytes.



# Модуль struct

**struct** - модуль для работы с байтовыми структурами в модуле.

Основные функции

- ▶ **pack**(формат, кортеж) -> строка байт
- ▶ **unpack**(формат, строка-байт) -> кортеж



# Модуль struct

Формат:

- ▶ b - знаковый один байт
- ▶ B - беззнаковый байт
- ▶ h - знаковое короткое целое число, 2 байта
- ▶ H - беззнаковое короткое целое число, 2 байта
- ▶ i - знаковое целое число, 4 байта
- ▶ I - беззнаковое целое число, 4 байта
- ▶ l - знаковое длинное целое число, 4 байта
- ▶ L - беззнаковое длинное целое число, 4 байта
- ▶ Q - беззнаковое очень длинное целое число, 8 байт
- ▶ f - число с плавающей точкой, 4 байта
- ▶ d - число с плавающей точкой двойной точности, 8 байт
- ▶ s - символы, count символов





# Модуль struct

```
from struct import pack, unpack

# Вещественное число -> строка байт
bs = pack('f', 3.1415) #  \x56\x0e\x49\x64

# строка байт -> вещественное число
x = unpack('f', bs)
```



## Модуль struct

```
from struct import pack, unpack

# Вещественное число -> строка байт
bs = pack('f', 3.1415) # \x56\x0e\x49\x64

# строка байт -> вещественное число
x = unpack('f', bs)
```

Конвертировать строи в список байт лучше с использованием метода `encode()`.



## Модуль struct

Преобразование набора данных в последовательность байт

```
# целое(4 байта), вещественное число(4 байта),  
# строка(3 байта)  
# -> последовательность байт
```

```
bs = pack('lf3s', 42, 3.1415, "abc".encode())
```

```
# обратное преобразование  
i,x,s = unpack('lf3s', bs)
```

```
# s нужно преобразовать из последовательности байт  
# в обычную строку  
s = s.decode()
```

Конвертировать строки в список байт лучше с использованием метода encode().



# Outline

Прошлые темы

Введение

Основные операции по работе с файлами

Файлы. Текстовый файлы  
Comma-Separated Values

Менеджеры контекста

Типы bytes и bytearray

Работа с файловой системой



# Работа с файловой системой

```
import os
```

## Некоторые функции

- ▶ `os.getcwd()` -> str, текущий каталог
- ▶ `os.chdir(путь)` - сменить текущий каталог
- ▶ `os.listdir(каталог)` -> список имён в каталоге  
по умолчанию *каталог* = '.', т.е. текущий каталог



# Работа с файловой системой

- ▶ `os.path.exists( путь )` -> True, если путь существует
- ▶ `os.path.isfile( путь )` -> True, если файл
- ▶ `os.path.isdir( путь )` -> True, если каталог



# Работа с файловой системой

- ▶ `os.remove( путь )` - удаляет файл
- ▶ `os.rmdir( путь )` - удаляет каталог



# Работа с файловой системой

- ▶ `os.path.getsize( путь-к-файлу )` -> размер файла в байтах
- ▶ `os.stat( путь )` -> информация о файле или каталоге





# Ссылки и литература

Ссылка на слайды

[github.com/VetrovSV/Programming](https://github.com/VetrovSV/Programming)

